Extrapolation and Generation of Benchmarks from Communication Traces

Frank Mueller, Xiaosong Ma
North Carolina State University

U.S. DEPARTMENT OF ENERGY

Agenda

- Overview of ScalaIOTrace
- Probabilistic Tracing and Replay (ICPP'11)
- Communication Extrapolation of Traces (PPoPP'11)
- Generation of Executable Specifications from Traces (ICS'11)
- Automatic Benchmark Code Generation from Traces

Introduction

- Contemporary HPC Systems
  - Size > 1000 processors
  - take IBM Blue Gene: ~74k nodes, ~300k cores
- Challenges on HPC Systems (large-scale scientific applications)
  - Communication & I/O scaling (MPI)
  - Communication & I/O analysis
- Procurements require performance prediction

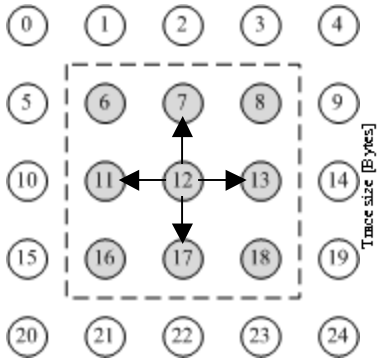
Communication Analysis

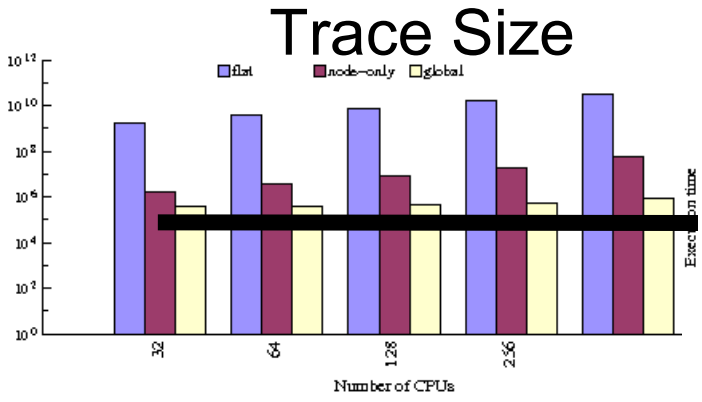
Existing approaches and short comings

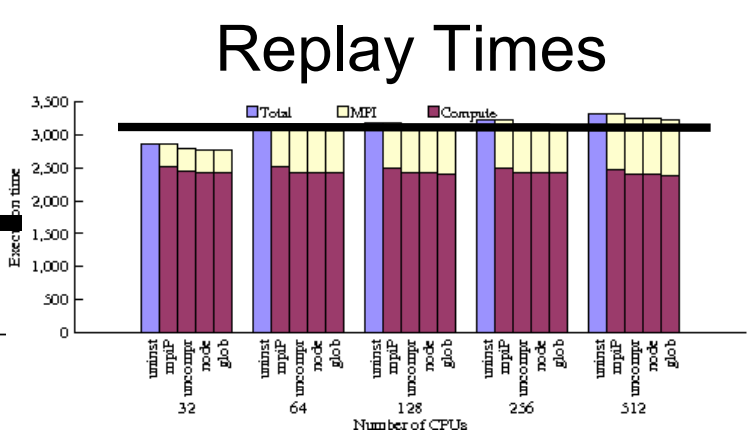
- Source code analysis
  - + Does not require machine time
  - - abstr. level: apps complex, no dyn. info
- Lightweight statistical analysis (mpiP)
  - + Low instrumentation cost
  - - only aggregate information
- Slicing
  - + Fast
  - - lacks temp. info
- Performance Modeling
  - + Abstract, somewhat general
  - - only high-level stats, arch.-dependent
- Fully captured (lossless): Vampir, VNG
  - + Full trace available for offline analysis
  - - not scalable (n traces) / need viz cluster

ScalaTrace: Lossless & Scalable Tracing

- Traces communication + I/O of MPI codes via interpositioning
- Trace size: Near constant size, one file represents all nodes
  - Intra-node (loop) & inter-node (task ID) compression of SPMD codes → preserves program structure
  - Location-independent encoding → scales
    - E.g., <10, MPI_Irecv(LEFT), MPI_Isend(RIGHT)>
  - Communication group encoding
    - <dim start_rank iteration_length stride {iteration_length stride}>
- Preserves timing for computation & communication (histograms)

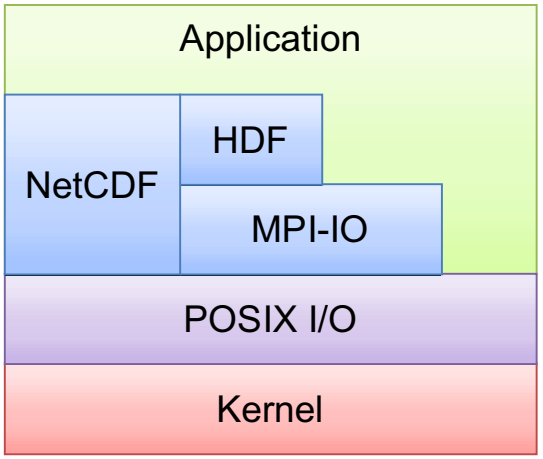






ScalaIOTrace: Multi-level I/O Capture (ICPP'11)

- Collect IO-traces at
  - MPI-IO layer
    → PMPI / Umpire (library)
  - POSIX I/O
    → gcc interposition (recompile)
- Replay I/O
  - Ability to replay MPI-IO calls
- Automated Trace Analysis
  - Customized replay
  - Adds generic event handlers for trace analysis
  - Future: Off-line analysis w/o replay (on your laptop)

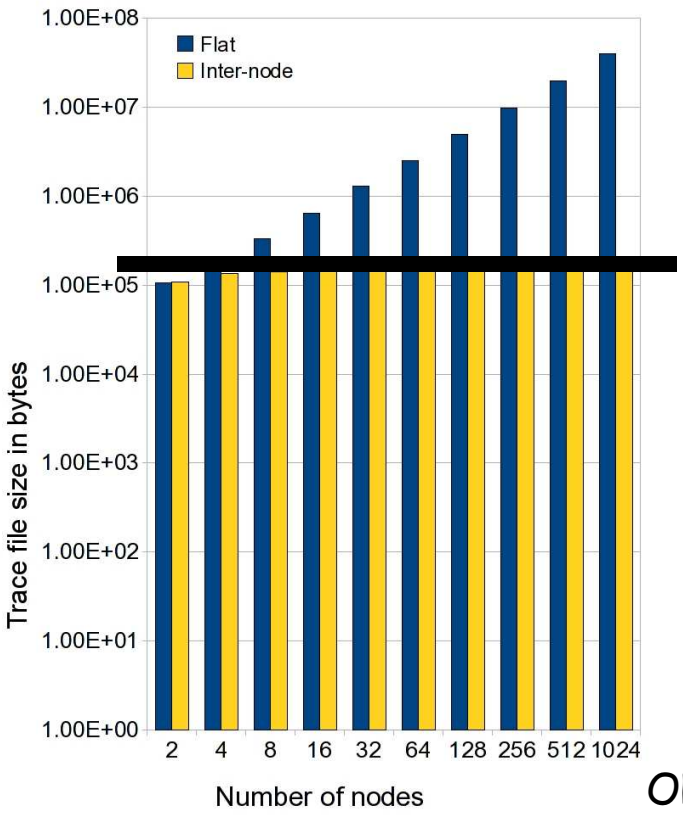


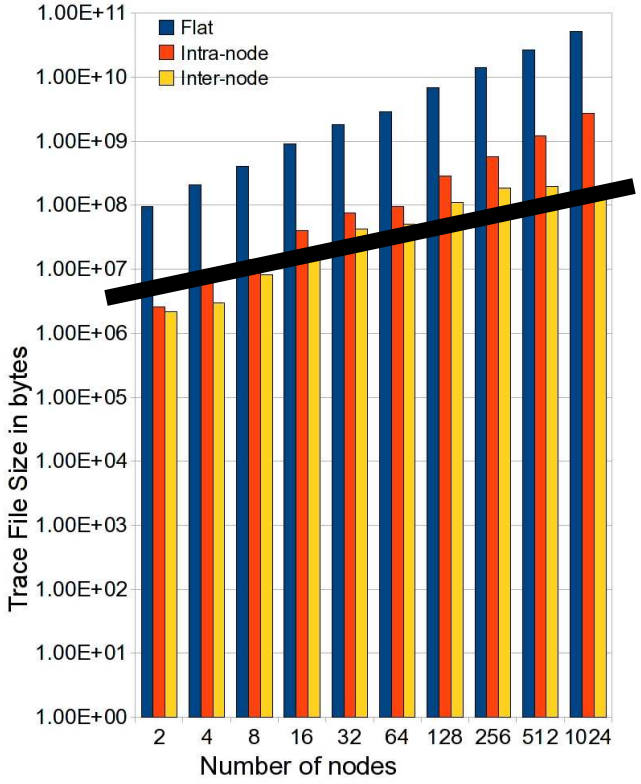
I/O Trace Collection

- Use ScalaTrace for lossless I/O tracing
- Collect MPI-IO (higher level) and POSIX I/O (lower level) traces
  - Expose multiple layers
  - Enable analysis of multi-level traces in a scalable way
- Scalable I/O Tracing:
  - File name compression
  - collective I/O → **file offsets,** patterns <start, stride, #elem>
  - file handles, custom data types → opaque pointers (buffered)
- MPI I/O: PMPI wrappers, POSIX I/O: Gnu –wrap

ScalaIOTrace and Replay: Trace Sizes

FlashIO: Perfect compression (HDF5) **POP: 2 orders magn. less (NetCDF)**





ORNL Jaguar

- Next slide: user-tunable precision knob
  - trade-off compression vs. accuracy
  - addresses data-dependent convergence points (POP, …)

Histogram Based Trace Collection

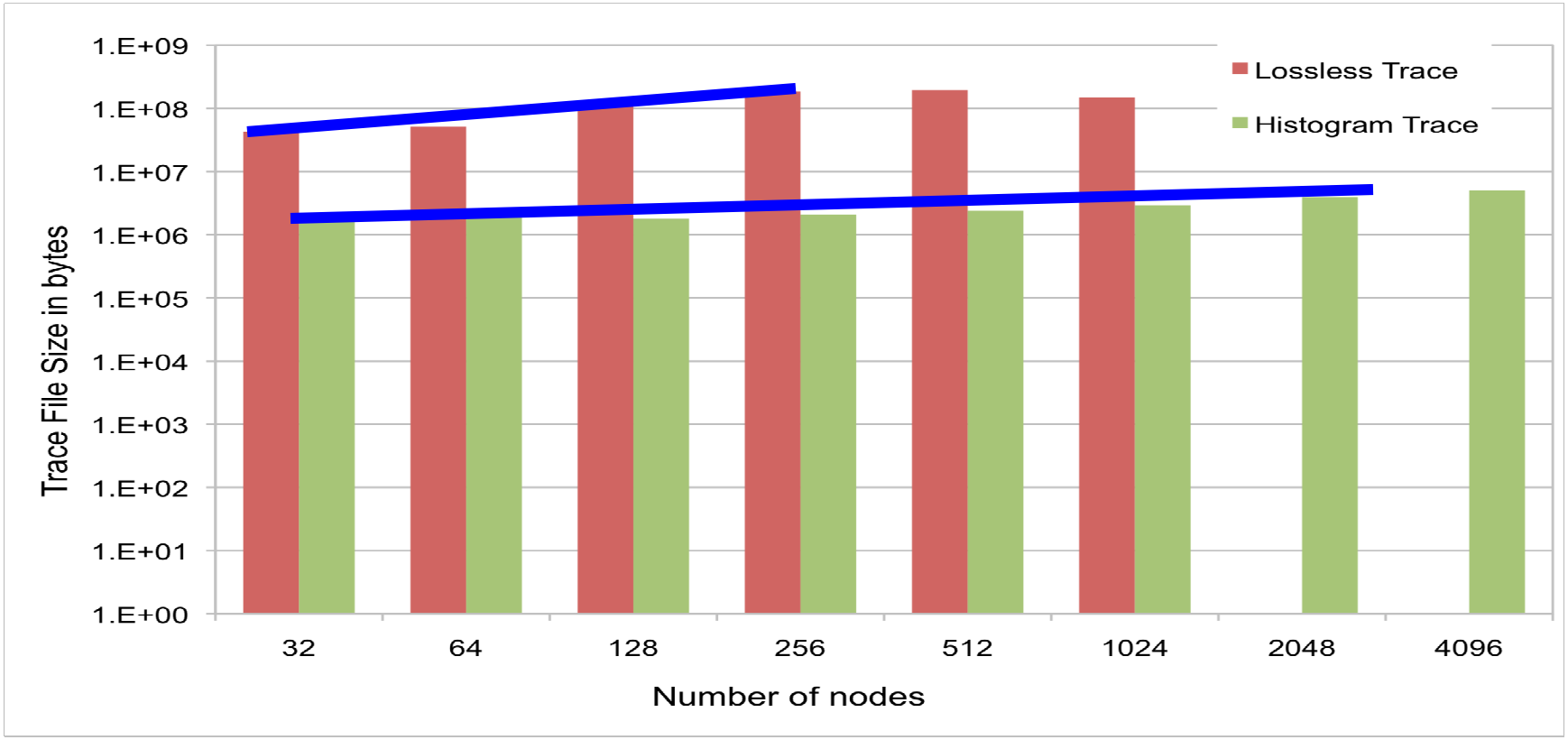
- Problem: Parameter mismatch → no compression → not scalable
- Solution: Histograms of multiple bins → lossy but scalable
- target precision level [%] → user specified
- If events match → collect lossless traces
- Else if difference of values < precision level → histogram
  - Bin value ranges dynamically adjusted

Deterministic Trace Replay

- Replay → "what-if" analysis, check correctness, …
- Lossless Trace Replay
  - Each node: reads its own events, reissue MPI&I/O calls
  - No trace decompression, dummy data (msgs, file data), Δ time
- Challenges in histogram-based trace replay
  - Uncoordinated selection of statistical comm. end-points/trace volume can lead to deadlock
  - Needs distributed, orchestrated replay
  - All nodes: read entire trace events, agree on values
  - E.g. Send Op: Sender calls Send(), receiver calls Irecv()

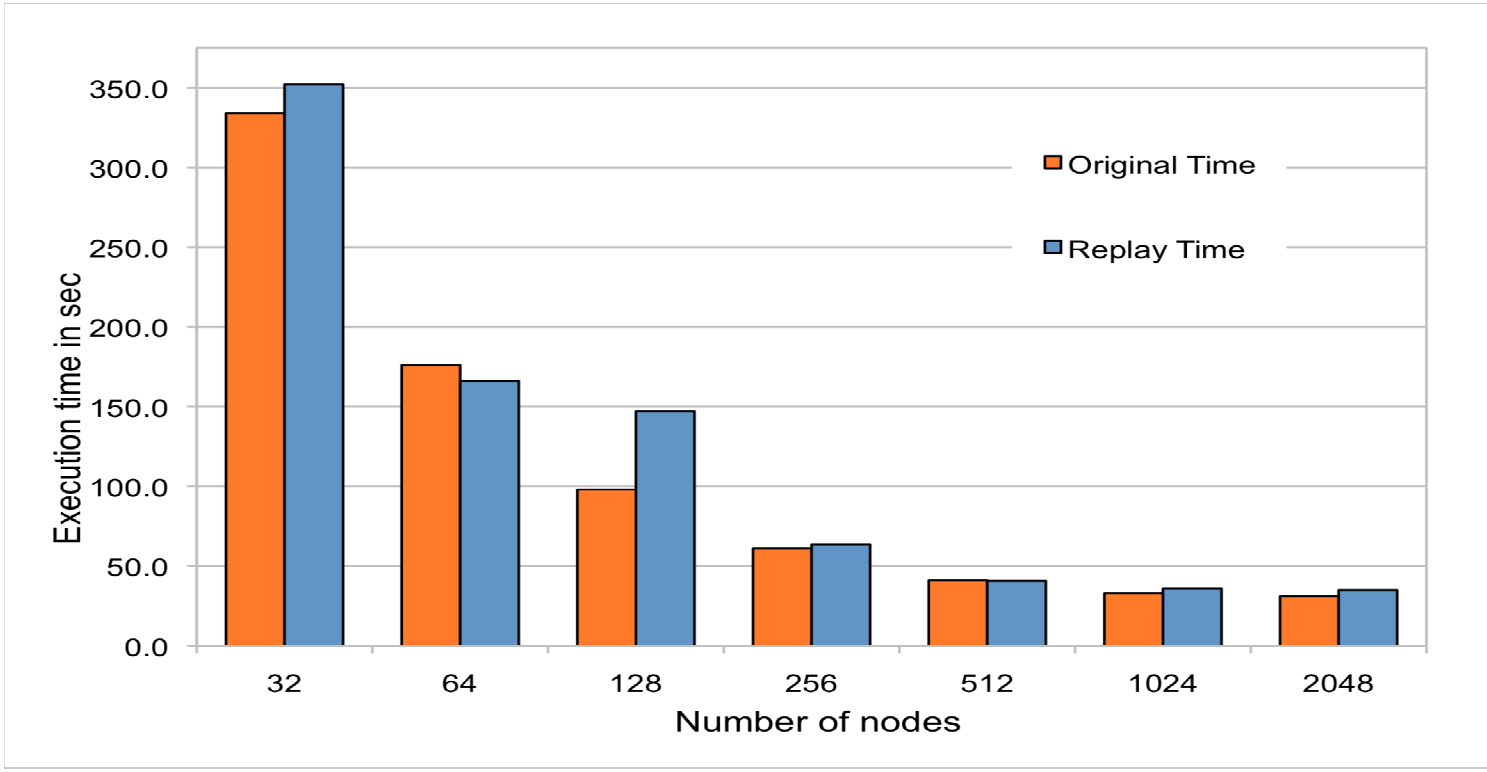
POP – Histogram-based trace size

- Histogram based traces:
  - Only sub-linear increase in trace size
  - 2 orders of magnitude smaller than lossless trace



POP – Histogram-based Trace Replay

- Replay time within 5% of original time till 512 nodes (except 128)
- Replay time within 12% for 1024/2048



Scalable Comm. & I/O Tracing (Summary)

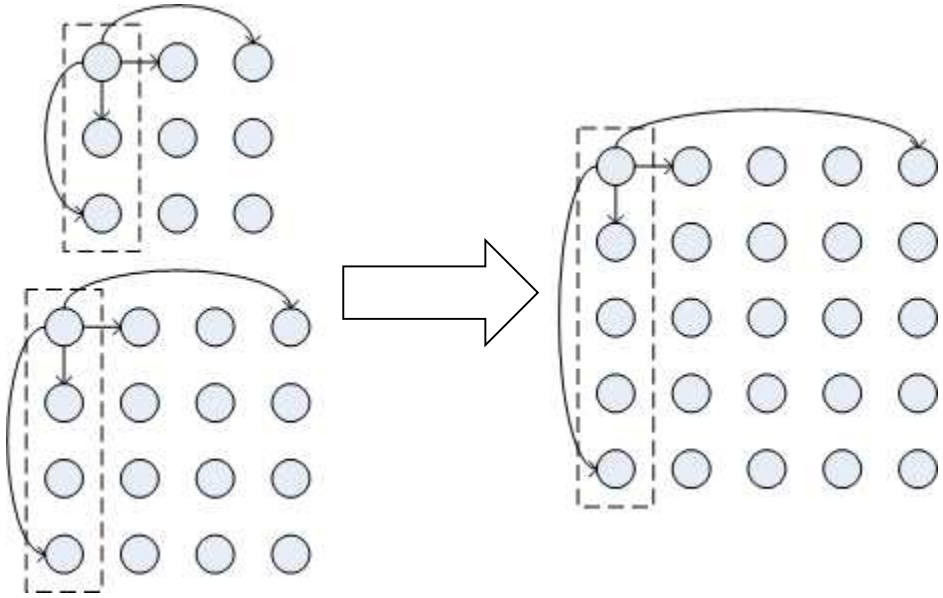
- Aggressive trace compression
  - Near constant size trace file for Flash I/O, CG
  - Only sub-linear increase for POP
- Capability to record traces at several layers
- Replay of histogram traces within 10% -15% of orig. time
- Framework for post-mortem trace analysis

NC STATE UNIVERSITY
Department of Computer Science

OAK RIDGE NATIONAL LABORATORY
Managed by UT-Battelle for the Department of Energy

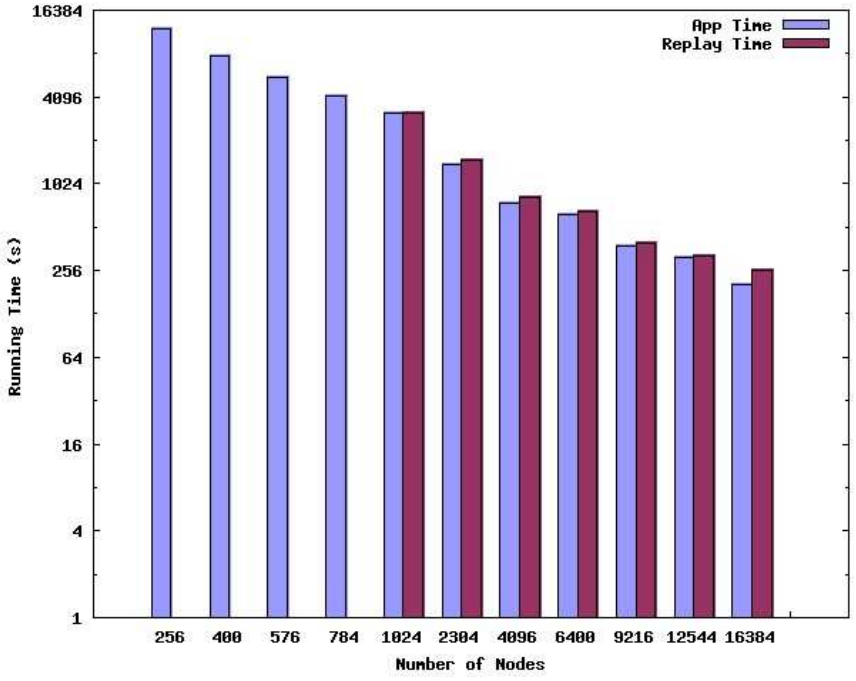
Communication Extrapolation (PPoPP'11)

- Motivation
  - Communication analysis at scale – without running app!
  - Modeling for procurements
  - Extrapolation on a single workstation!
- Idea: synthetically generate communication traces:
  - k small traces from app → large traces
  - E.g., P=8,16,32,64 nodes trace → P=4096 trace (or any P)
- Replay large trace/analyze it
- Challenges:
  - Topology detection
    - Meshes/stencils
  - Message payloads
    - Gaussian elimination
  - Time extrapolation
    - Curve fitting

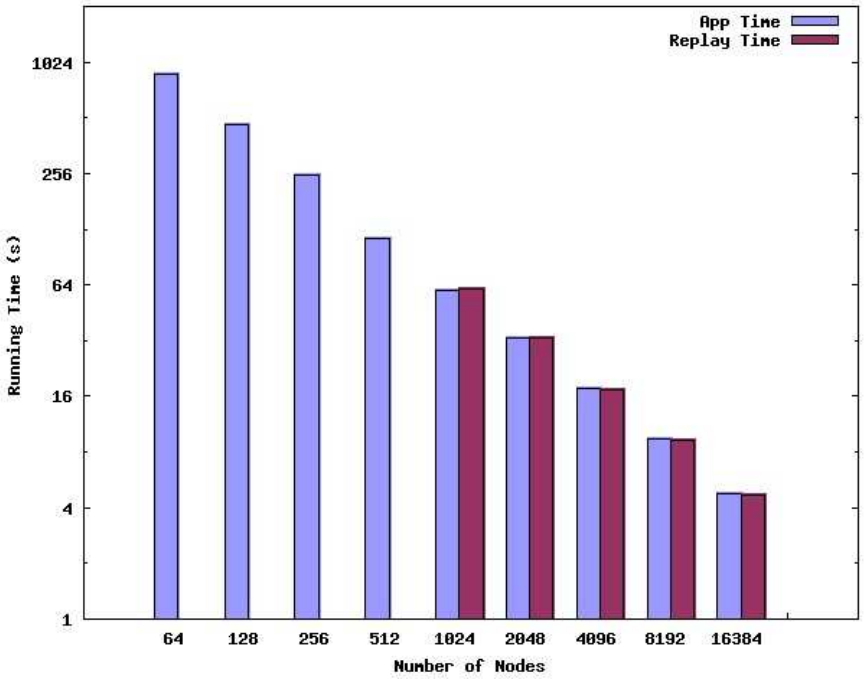


Accuracy of Timing Extrapolation (i)

- for NPB: BT, EP, FT, IS, CG
- up to 16k nodes (not cores)
- t(extrapolated replay) = t(app)
- Accuracy = |Replay Time - App Time|/App Time) → generally>90%



BT (class E)



FT (class D)

Extrapolation (Summary)

Contributions:

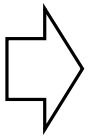
- Algorithms & techniques for comm. extrapolation, handles
  - trace events
  - execution times
- Based on app runs at smaller scale
- Extrapolation shown to be
  - correct
  - accurate
- Obtains comm. behavior of parallel app at arbitrary scale
  - without actual execution at this scale → unprecedented
- Future Work:
  - Weak scaling

Extrapolation not so elusive anymore

Communication Benchmark Generation (ICS'11)

- Goal: Generate comm. benchmarks from apps that are
  - easy to 1) distribute, 2) use, 3) modify
- Extracted benchmarks from applications are
  - Performance-accurate
  - Application logic is stripped out
  - Readable, portable, modifiable
    - Collectives are consolidated
    - Nondeterminism has been eliminated
- Target coNCePTuaL: language for rapid generation of network benchmarks
- Compiler + runtime library

```
for(i = 0; i < 10; i++){
   MPI_Irecv(10, LEFT);
   MPI_Isend(10, RIGHT);
   MPI_Waitall();
}
```



```
For 10 repetitions {
   All tasks t asynchronously send a
      10-byte message to task t+1 then
   all tasks await completion
}
```

Code Generation from Application Trace

Application → Application Trace → Benchmark in coNCePTuaL

```
MPI_Init();
func1();
for(i = 0; i < 10; i++){
   func2();
   MPI_Irecv(LEFT);
   func3();
   MPI_Isend(RIGHT);
   func4();
   MPI_Waitall();
}
func5();
MPI_Barrier();
MPI_Finalize();
```

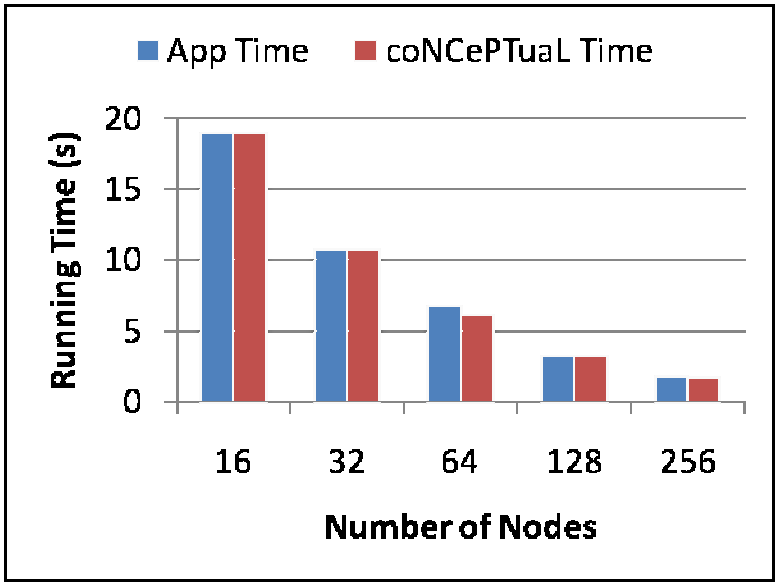
```
RSD1:   <All, Init>
        T1
PRSD1:  <10, RSD2, RSD3, RSD4>
        T2
RSD2:   <All, Irecv, LEFT>
        T3
RSD3:   <All, Isend, RIGHT>
        T4
RSD4:   <All, Waitall>
        T5
RSD5:   <All, Barrier>
RSD6:   <All, Finalize>
```

```
All tasks compute for T1 seconds
For 10 repetitions{
   All tasks compute for T2 seconds
   All tasks t asynchronously receive an x-byte
         message from task t-1 then
   All tasks compute for T3 seconds
   All tasks t asynchronously send an x-byte
         message to unsuspecting task t+1 then
   All tasks compute for T4 seconds
   All tasks await completion
}
All tasks compute for T5 seconds
All tasks synchronize
```

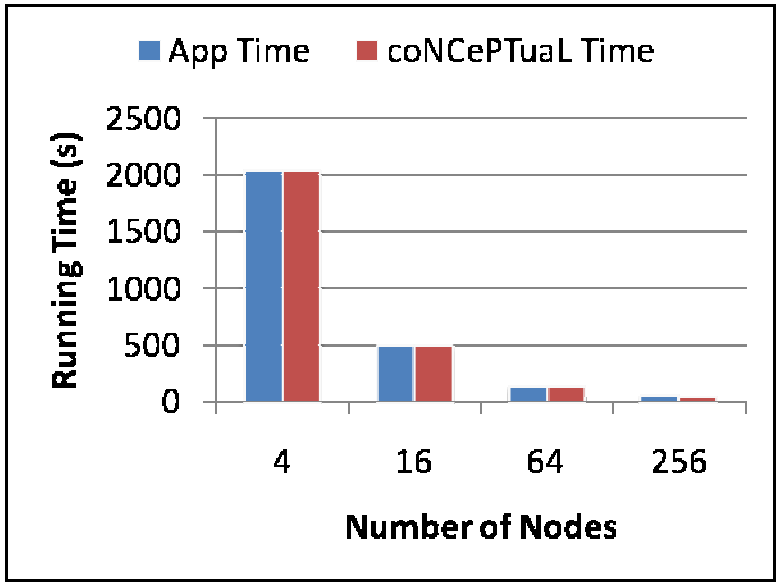
- All comm. events & computation times generated
- Benchmark contains loop structure
  → easier to read/modify than translation to straight-line code
- Contributions: consolidate collectives, eliminate non-determinism

Accuracy of Generated Timings

- Time the application and the generated benchmark, and compare the results
- Mean absolute percentage error is only 2.9% → formula: |TcoNCePTuaL – Tapp|/Tapp x 100



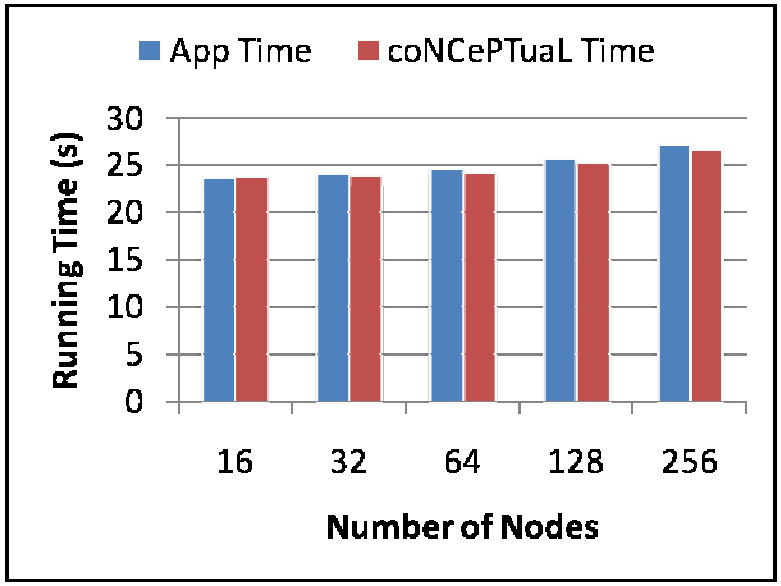
IS



LU

Accuracy of Generated Timings

- Time the application and the generated benchmark, and compare the results
- Mean absolute percentage error is only 2.9% → formula: |TcoNCePTuaL – Tapp|/Tapp x 100

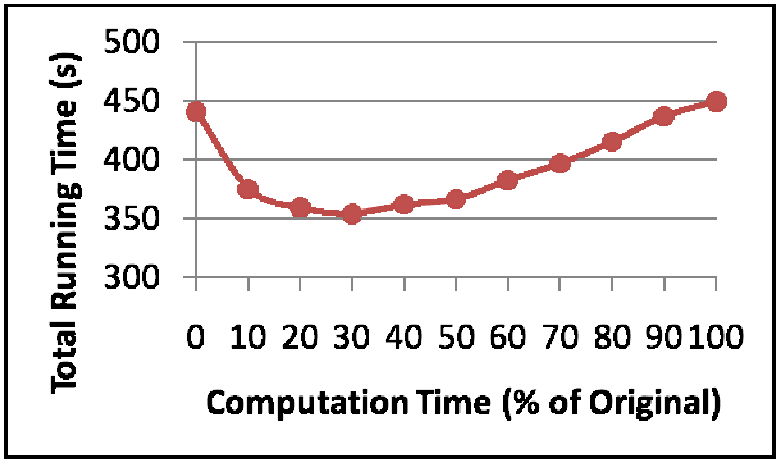


Sweep3D: Weak Scaling

Applications of the Benchmark Generator

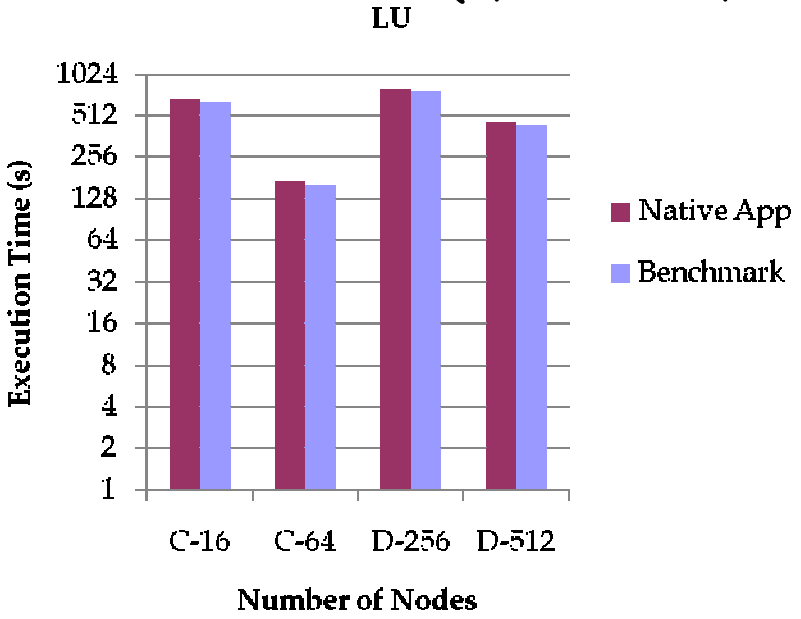
- Determine limits of computation/comm. overlap or effect of computational acceleration (e.g., GPUs)
  - Experiment: ARC cluster, 64 cores, Ethernet, BT b'mark
  - Shorten the spin times gradually
    - 100%: original compute overhead (simulated w/ spin)
    - 0%: no compute overhead → infinitely fast processor

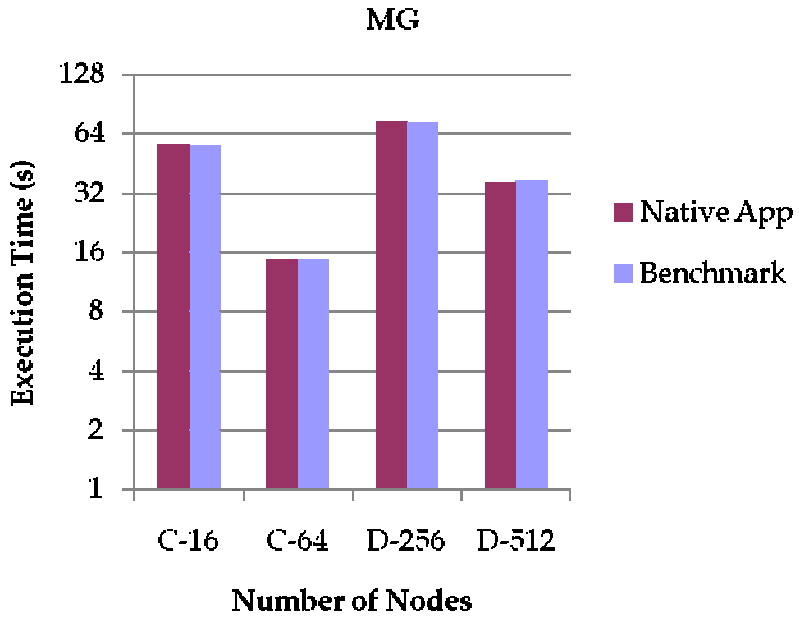
- Best speedup: ~3X → overall runtime reduced by 22%
- 0%: network contention or extra memory copies
- Platform-specific result



(3) C Benchmark Generation Results – Timing Accuracy

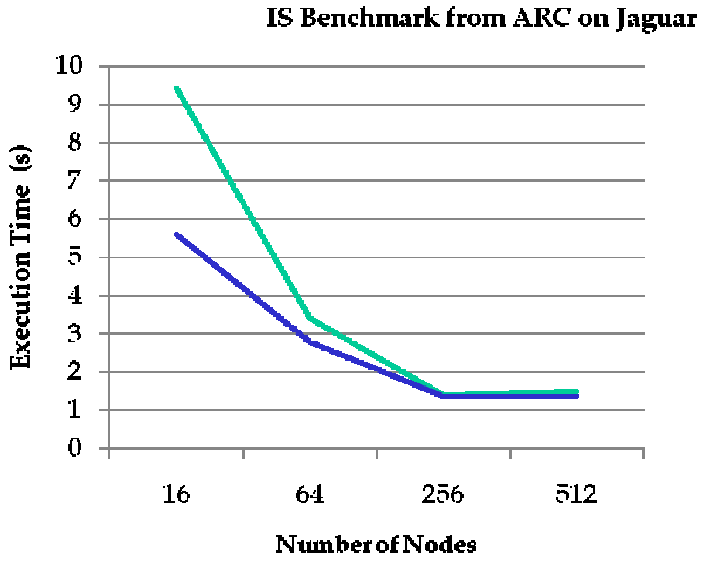
- time from Init → Finalize for app and benchmark
  |Tgen – Tapp|/Tapp x 100
- Timing accuracy = ~ 6.7% (avg. error)
- ARC cluster (1,7k cores, 16 cores/node Opteron, 32 GB RAM)

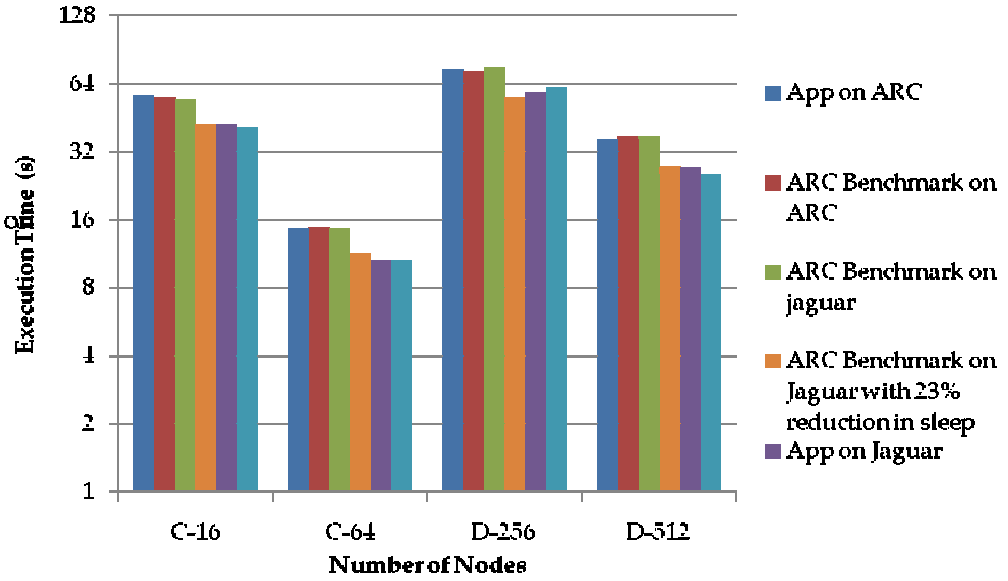




Cross Platform Results: ARC vs. Jaguar

- Jaguar ~23% faster execution on compute kernels
- Resemblance to benchmark obtained on ARC
  - IS: strong scaling reduced per node work → close match @ 256 tasks
  - Nearly perfect match after 23% speed correction





Summary

- Scalable Comm & I/O tracing is realistic
- Trace extrapolation feasible → exascale modeling
- An automatic communication benchmark generation framework
  - ScalaTrace → coNCePTual or C
- Generate benchmarks from real-world apps: comm. & I/O
  - Ensure performance fidelity, abstract away application logic
  - Readable, portable, modifiable, reproducible
    - -Consolidation of collectives
    - -Elimination of nondeterminism
  - Obfuscates code → for restricted source code access
  - Facilitate "what-if" analysis

Acknowledgements

- Xing Wu (NCSU, intern @ LANL)
- Scott Pakin (LANL)
- Karthik Virjayakumar (NCSU, intern @ ORNL)
- Phil Roth (ORNL)
- Mike Noeth, Prasun Ratn (NCSU, interns @ LLNL)
- Martin Schulz, Bronis R. de Supinski (LLNL)

- best paper [IPDPS'07,JPDC], timed replay [ICS'08], I/O [PDSW'09], extrap [PPoPP'11], gen. specs [ICS'11], prob. Replay [ICPP'11]
- Code availabile under BSD license:

  moss.csc.ncsu.edu/~mueller/ScalaTrace

- Funded in part by NSF 0937908, 0429653, 0410203, CAREER 0237570, 0958311
- Part of work … auspices … U.S. DoE by UC-LLNL under contract No. W-7405-Eng-48 + UT-Batelle, LLC DE-AC05-00OR22725 + Sandia DE-AC52-06NA25396.